Homework 1 prepared for Prof. M. Khalid Jawed teaching MAE 263F in Fall 2025

J. Dekker

I. Task

This homework required two main tasks. The first of which is to plot a simulation of the spring network shown in Figure 1 at 0 seconds, 0.1 seconds, 1 second, 10 seconds and 100 seconds. These plots can be seen in Figure 2 to Figure 6.

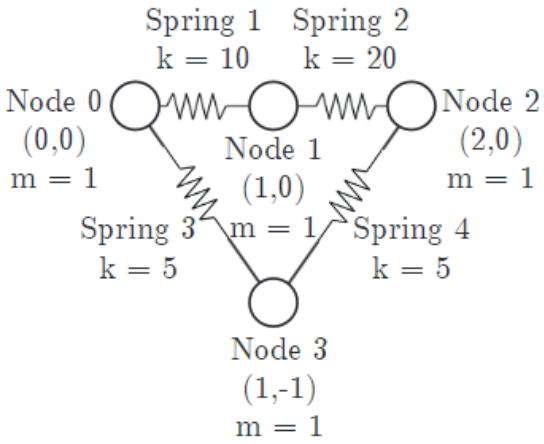

Figure 1: Simulated spring network.

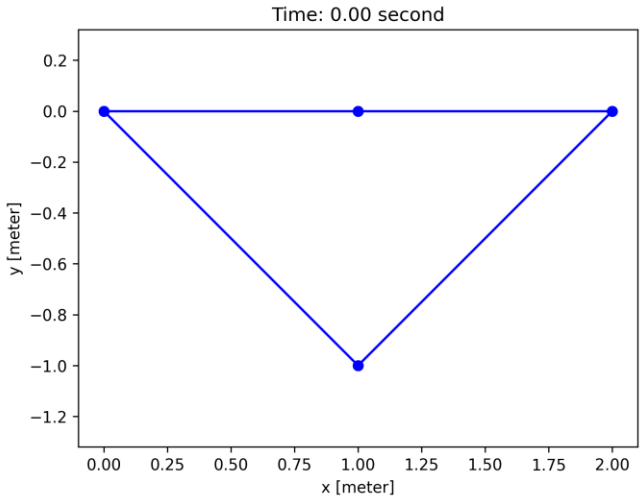

Figure 2: Spring network positions at Time = 0.0s

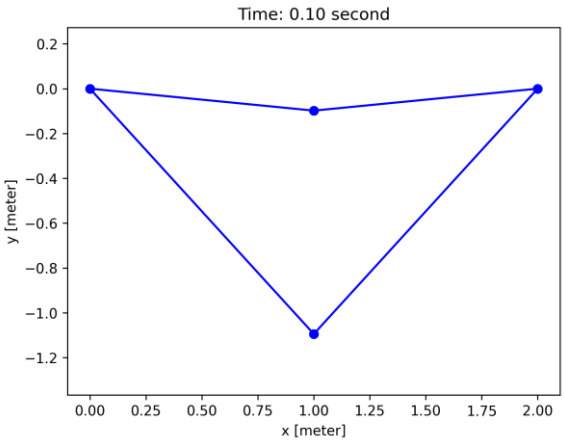

Figure 3: Spring network positions at Time = 0.1s

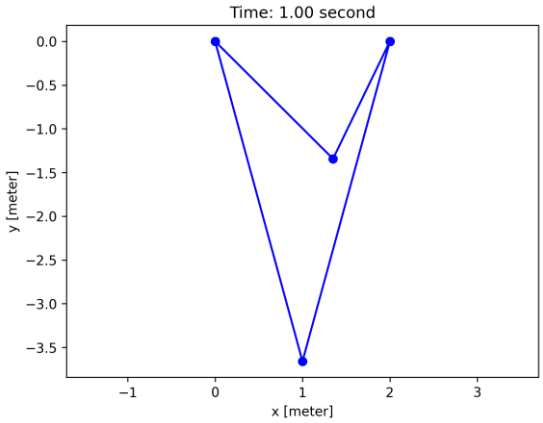

Figure 4: Spring network positions at Time = 1.0s

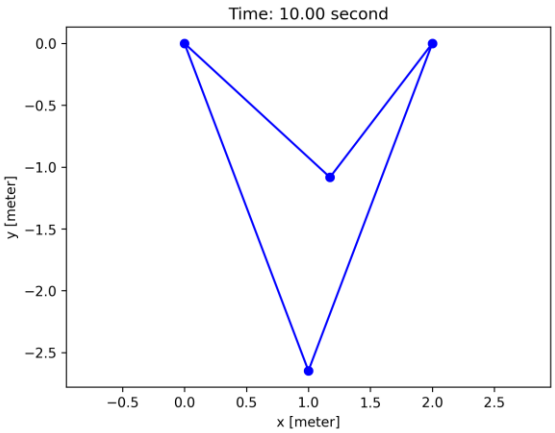

Figure 5: Spring network positions at Time = 10.0s

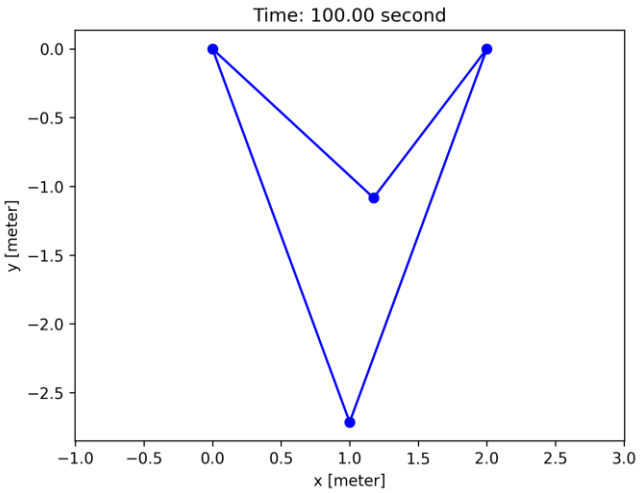

Figure 6: Spring network positions at Time = 100.0s

The second task is to plot the movement of the two free nodes (Nodes 1 and 3 seen in Figure 1), which is seen in Figure 7.

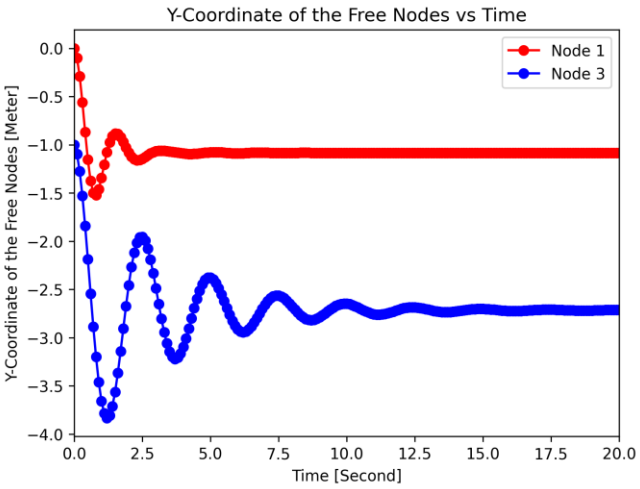

Figure 7: Y-Coordinates of the free nodes in the spring system.

II. Discussion

I. Pseudocode and code structure:

The code performs multiple steps before starting the simulation. These initializing steps are summarized into:

1) Import the spring network.
2) Initialize the main variables.
3) Define the free DOFs.

With that complete the simulation begins by calling a solver, solverNewtonRaphson, to determine the next position and velocity based on the current time step data. Determining the new time step data is done iteratively using the Newton-Raphson Method. Each iteration of the solver requires a new force vector and Jacobian. Therefore, a separate function called getForceJacobianImplicit was created for this task.

Broadly speaking, the Equation of Motion of the system has three terms: inertia, energy, and external forces. To calculate the force vector and Jacobian of the inertia and energy terms require a guess of the next position. Also note that the gradient of energy is the force and the hessian of energy is a Jacobian, therefore the functions gradEs and hessEs are used. Lastly the external forces in this simulation are only gravity, therefore they are only dependent on the mass of each node. This is captured in a function called getExternalForce for clarity. All this is summed up and returned into the solver.

The solver will iterate on the guesses of the new state while only updating the positions of the free degrees of freedom. Once the error is sufficiently low, the solver returns the positions and velocities of the next state.

As the main script continues finding new positions, it will plot certain time markers as needed. This function called plot takes in the current positions of the nodes, the time, and index matrix for the springs. It plots the pair of nodes which correspond to a spring within the network.

Figure 8 shows the flow of the main script from left to right with all the function calls and where they lie nested within themselves and for loops.

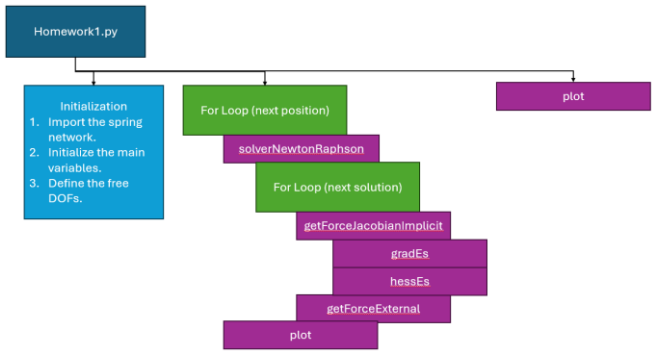

Figure 8: Block diagram of the main solving script.

II. How do you choose an appropriate time step size Δt?

With or without damping present in the system we expect a certain level of oscillation in the system at a steady state. The time step must be smaller than the period of this oscillation to best capture the behavior of the system.

Therefore, selecting a time step is truly user dependent on the number of steps used to define a sine wave. The time step would then be the approximate period of the spring-mass system divided by the number of desired points.

III. Simulate the spring network using Explicit Euler for t ∈ [0, 1] s. If it becomes unstable, reduce Δt and try again. If it still fails even at Δt = 10−6 s, state this in the report. Explain which method (implicit vs. explicit) is preferable for this spring network and why.

There was a separate explicit solver made to replace the implicit Newton-Raphson solver used in the main iteration look. Along with changing the solver, the time step and max time variables were adjusted the parameters of this question.

Even at a time step of 10^{-6} seconds the explicit solver did not converge on a solution but continued into infinity as seen in Figure 9.

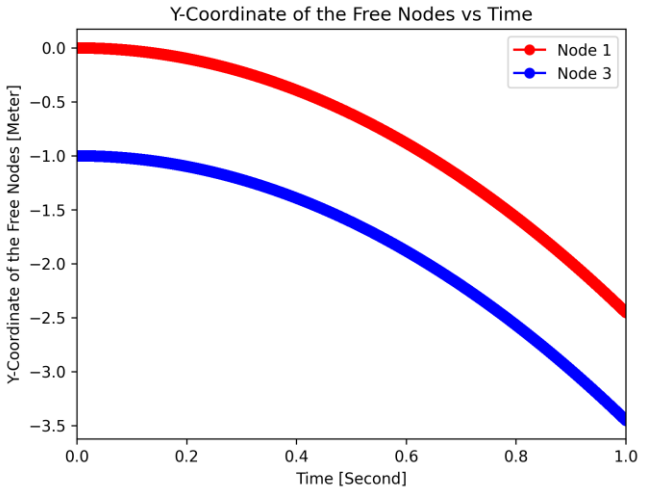

Figure 9: Y-Coordinates of the Free Nodes using an Explicit Solver.

The implicit solver is therefore preferable as it parallels the system better than the explicit solver. However, the implicit solver is not perfect since it is showing damping in the system which is not present.

IV. The simulation with implicit Euler appears to reach a static state (numerical damping). Read about the Newmark–β family of time integrators and explain how such integrators can mitigate such artificial damping.

Implicit Euler solvers see artificial damping because it assumes the future time-step is stable. This inherently settles down the system's behavior. Whereas an explicit solver has the potential to overshoot the true behavior of the system and simulate the object moving into infinity. The Newmark-β solvers balance between implicit and explicit solvers. Using the β and γ parameters the damping in the solver can be tuned for the desired result.